Capstone Project - Credit Card Default Prediction

Data scientist Nanodegree

1. **Project Overview**

Credit default is important for credit card bank to manage users. In this project, I use a dataset detailing personal and payment information of credit card owners to predict whether someone will default in the future. The dataset is from a foreign bank that experiences a high level of credit card defaults.

2. **Problem Statement**

The goal of this project is to use a dataset detailing personal and payment information of credit card owners to predict whether someone will default in the future. I explored the dataset and applied 4 different models to build a binary classifier in Python.

In this report, I first describe the dataset, how I preprocess and analyze it. Then, I use hyperparameter optimization to configure the best model from logistic regression, decision tree, neural network, and random forest. At the end, I test the results by 70/30 training/validation split.

3. Metrics

Accuracy is a common metric for binary classifiers; it takes into account both true positive and true negatives with equal weight.

$$accuracy = \frac{true\ positives + true\ negatives}{dataset\ size}$$

Precision - Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. The question that this metric answer is of all passengers that labeled as survived, how many actually survived? High precision relates to the low false positive rate. We have got 0.788 precision which is pretty good.

Precision = TP/TP+FP

Recall (Sensitivity) - Recall is the ratio of correctly predicted positive observations to the all observations in actual class - yes. The question recall answers is: Of all the passengers that truly survived, how many did we label? We have got recall of 0.631 which is good for this model as it’s above 0.5.

Recall = TP/TP+FN

F1 score - F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. Intuitively it is not as easy to understand as accuracy, but F1 is usually more useful than accuracy, especially if you have an uneven class distribution. Accuracy works best if false positives and false negatives have similar cost. If the cost of false positives and false negatives are very different, it’s better to look at both Precision and Recall. In our case, F1 score is 0.701.

F1 Score = 2*(Recall * Precision) / (Recall + Precision)

4. Analysis

4.1 Data Exploration

The credit Card default dataset contains 30000 rows and 32 columns, with the first column as customer ID. A binary variable, account default (1=yes, 0=no) in the second column is used here as the target. Therefore, there are 30000 observations with 30 input attributes, including 1 binary, 3 nominal, and 26 interval attributes. The data dictionary is shown in Figure 1.

ATTRIBUTE	Type	DESCRIPTION
Default	Binary	Account Default? (1=yes, 0=no)
Gender	Binary	Male or Female? (1=female, 2=male)
Education	Nominal	Education level :0, 1, 2, 3, 4, 5, 6
Marital_Status	Nominal	Martial Status: 0, 1, 2, 3
Card_Class	Nominal	Class 1, 2 or 3
Age	Interval	Age from 20 to 80
Credit_Limit	Interval	Credit limit from 100 to 80,000
Jun_Status	Interval	Months Behind Payment : from -2 to +8
May_Status	Interval	Months Behind Payment: from -2 to +8
Apr_Status	Interval	Months Behind Payment: from -2 to +8
Mar_Status	Interval	Months Behind Payment: from -2 to +8
Feb_Status	Interval	Months Behind Payment: from -2 to +8
Jan_Status	Interval	Months Behind Payment: from -2 to +8
Jun_Bill	Interval	Monthly Bill: -12,000 to +32,000
May_Bill	Interval	Monthly Bill: -12,000 to +32,000
Apr_Bill	Interval	Monthly Bill: -12,000 to +32,000
Mar_Bill	Interval	Monthly Bill: -12,000 to +32,000
Feb_Bill	Interval	Monthly Bill: -12,000 to +32,000
Jan_Bill	Interval	Monthly Bill: -12,000 to +32,000
Jun_ Payment	Interval	Payment for the month: 0 to 60,000
May_ Payment	Interval	Payment for the month: 0 to 60,000
Apr_ Payment	Interval	Payment for the month: 0 to 60,000
Mar_ Payment	Interval	Payment for the month: 0 to 60,000
Feb_Payment	Interval	Payment for the month: 0 to 60,000
Jan_ Payment	Interval	Payment for the month: 0 to 60,000
Jun_ PayPercent	Interval	Ratio - f(payment/bill) from 0 to 1
May_ PayPercent	Interval	Ratio - f(payment/bill) from 0 to 1
Apr_ PayPercent	Interval	Ratio - f(payment/bill) from 0 to 1
Mar_ PayPercent	Interval	Ratio - f(payment/bill) from 0 to 1
Feb_PayPercent	Interval	Ratio - f(payment/bill) from 0 to 1
Jan_PayPercent	Interval	Ratio - f(payment/bill) from 0 to 1

Figure 1. Data dictionary

4.2 Exploratory Visualization

The plot below shows the distribution of credit default in dataset. As we can see from the Figure 2, the dataset is highly skewed; most users are credit default. This will help us to set parameter of our model.

```
count    30000.000000
mean         0.161800
std          0.368273
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max          1.000000
Name: Default, dtype: float64
mode:  0    0
dtype: int64
==========================================================================
=====
0    25146
1     4854
Name: Default, dtype: int64
```

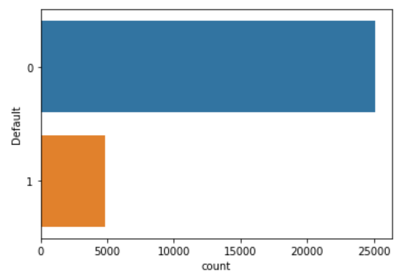

Figure 2. Credit Default Distribution

Fig. 3 The following plot shows the relationship between gender and credit default. This information can be helpful when we validate the influence of feature toward the response variable. As we can see, the male (Gender = 2) takes large proportion in the dataset and female (Gender = 1) is more prefer not set credit default.

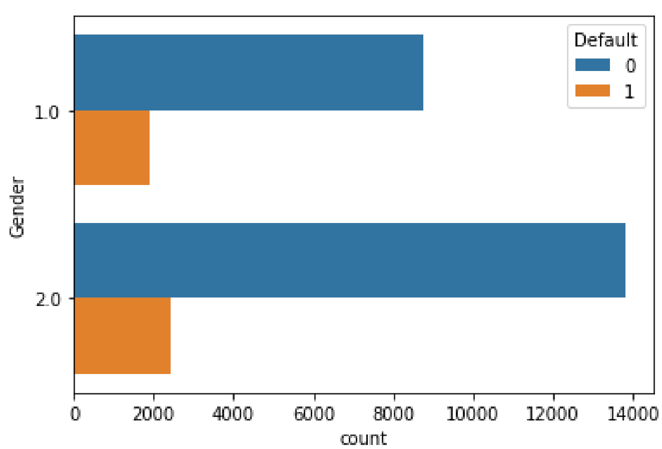

Figure 3. Credit Default Distribution by Gender

5. **Model**

5.1 Preprocessing

Data preprocessing is first needed for applying following machine learning models. There are some missing values and outliers in the dataset. I first identify the outliers, replace them with missing value.

In Python, i first created a data map to describe the attributes in Credit Card data, their level (interval, binary or nominal), and the characteristics of the attributes, in which the 0 represents the interval, the 1 represents binary and 2 represents the nominal. Also, I impute the lower limit and upper limit for the interval attributes and categories for the binary and nominal attributes. The data map is shown in Figure 4:

```
attribute_map = {
    'Default':[1,(1,0),[0,0]],
    'Gender':[1,(1,2),[0,0]],
    'Education':[2,(0,1,2,3,4,5,6),[0,0]],
    'Marital_Status':[2,(0,1,2,3),[0,0]],
    'card_class':[2,(1,2,3),[0,0]],
    'Age':[0,(20,80),[0,0]],
    'Credit_Limit':[0,(100,80000),[0,0]],
    'Jun_Status':[0,(-2,8),[0,0]],
    'May_Status':[0,(-2,8),[0,0]],
    'Apr_Status':[0,(-2,8),[0,0]],
    'Mar_Status':[0,(-2,8),[0,0]],
    'Feb_Status':[0,(-2,8),[0,0]],
    'Jan_Status':[0,(-2,8),[0,0]],
    'Jun_Bill':[0,(-12000,32000),[0,0]],
    'May_Bill':[0,(-12000,32000),[0,0]],
    'Apr_Bill':[0,(-12000,32000),[0,0]],
    'Mar_Bill':[0,(-12000,32000),[0,0]],
    'Feb_Bill':[0,(-12000,32000),[0,0]],
    'Jan_Bill':[0,(-12000,32000),[0,0]],
    'Jun_Payment':[0,(0,60000),[0,0]],
    'May_Payment':[0,(0,60000),[0,0]],
    'Apr_Payment':[0,(0,60000),[0,0]],
    'Mar_Payment':[0,(0,60000),[0,0]],
    'Feb_Payment':[0,(0,60000),[0,0]],
    'Jan_Payment':[0,(0,60000),[0,0]],
    'Jun_PayPercent':[0,(0,1),[0,0]],
    'May_PayPercent':[0,(0,1),[0,0]],
    'Apr_PayPercent':[0,(0,1),[0,0]],
    'Mar_PayPercent':[0,(0,1),[0,0]],
    'Feb_PayPercent':[0,(0,1),[0,0]],
    'Jan_PayPercent':[0,(0,1),[0,0]]}
```

Figure 4. Data map

After creating the data map, we use it with an algorithm to identify the outliers and missing values in each attribute. For interval variables, the value outside the limits will be marked as an outlier; for categorical variables (binary and nominal), the value that does not match the categories will be marked as an outlier.

Then, in this case, we set all the outliers as missing value and impute these values along with all other missing values. In Python, “mean” is used for interval attributes and “mode” for categorical ones in the ‘sklearn’ package.

Besides, the machine learning attributes in sklearn work best if the interval variables are scales. The StandardScaler method in the sklearn preprocessing package is used to do it. Next, one-hot (SAS) encoding is applied for nominal attributes in Python by sklearn OneHotEncoder. At last, we combined scaled imputed interval variables, one-hot encoded nominal variables and imputed binary variable together and drop the last column of the one-hot encoded nominal variables for logistic regression and the data frame without dropping last column for rest of solution model. The data frames are shown as following respectively,

```
Imputed & Scaled lgr DataFrame:
        Age  Credit_Limit  Jun_Status  May_Status  Apr_Status  Mar_Status  \
0 -1.391899     -1.131883    1.794564    1.782348   -0.696663   -0.666599
1 -1.149700     -0.366111   -0.874991    1.782348    0.138865    0.188746
2 -0.180901     -0.591338    0.014861    0.111736    0.138865    0.188746
3  0.182399     -0.906656    0.014861    0.111736    0.138865    0.188746
4  2.604397     -0.906656   -0.874991    0.111736   -0.696663    0.188746

   Feb_Status  Jan_Status  Jun_Bill  May_Bill  ...  Education3  Education4  \
0   -1.530046   -1.486041 -0.643742 -0.648829  ...         0.0         0.0
1    0.234917    1.992316 -0.660503 -0.668231  ...         0.0         0.0
2    0.234917    0.253137 -0.298915 -0.494888  ...         0.0         0.0
3    0.234917    0.253137 -0.057224 -0.012891  ...         0.0         0.0
4    0.234917    0.253137 -0.579693 -0.612646  ...         0.0         0.0

   Education5  Marital_Status0  Marital_Status1  Marital_Status2  card_class0  \
0         0.0              0.0              1.0              0.0          1.0
1         0.0              0.0              0.0              1.0          0.0
2         0.0              0.0              0.0              1.0          0.0
3         0.0              0.0              1.0              0.0          1.0
4         0.0              0.0              1.0              0.0          1.0

   card_class1  Default  Gender
0          0.0      1.0     2.0
1          1.0      0.0     2.0
2          1.0      0.0     2.0
```

Figure 5. The data frame for logistic regression

5.1.2 Logistic Regression

The response variable in this case is a binary variable. We can create logistic regression model for prediction. we use the Logistic Regression in sklearn.linear_model package to fit the logistic regression model for the whole data. Before the fitting process, we create the features (X) as the predictor variable and target (y) as the response variable for building logistic regression model. At last, we used the logre in Class_regression package to display the coefficients of all the variable and metrics in Figure 6 and Figure 7:

```
Logistic Regression Model using Entire Dataset

Coefficients:
Intercept......        -1.6860
Age............         0.0646
Credit_Limit...        -0.2967
Jun_Status.....         0.7535
May_Status.....         0.3222
Apr_Status.....         0.1002
Mar_Status.....         0.0978
Feb_Status.....         0.1419
Jan_Status.....         0.1726
Jun_Bill.......        -0.2220
May_Bill.......         0.3994
Apr_Bill.......        -0.1362
Mar_Bill.......         0.1415
Feb_Bill.......        -0.0504
Jan_Bill.......         0.0429
Jun_Payment....        -0.2446
May_Payment....        -0.1746
Apr_Payment....        -0.0686
Mar_Payment....        -0.0492
Feb_Payment....        -0.1571
Jan_Payment....        -0.0350
Jun_PayPercent.         0.2483
May_PayPercent.         0.1631
Apr_PayPercent.        -0.0018
Mar_PayPercent.         0.1200
Feb_PayPercent.         0.1878
Jan_PayPercent.         0.0613
Education0.....        -0.5484
Education1.....         0.2830
Education2.....         0.2176
Education3.....         0.1845
Education4.....        -0.6777
Education5.....        -0.8414
Education6.....        -1.6177
Marital_Status0        -0.0497
Marital_Status1        -0.2392
Marital_Status2        -0.2713
Marital_Status3        -0.3653
card_class0....        -0.0944
```

Figure 6. Coefficients of the logistic regression model

```
Model Metrics
Observations...............     30000
Coefficients...............        39
DF Error...................     29961
Mean Absolute Error........    0.2100
Avg Squared Error..........    0.1036
Accuracy...................    0.8622
Precision..................    0.6761
Recall (Sensitivity).......    0.2851
F1-Score...................    0.4011
MISC (Misclassification)...     13.8%
     class 0...............      2.6%
     class 1...............     71.5%


     Confusion
       Matrix      Class 0    Class 1
Class 0.....      24483       663
Class 1.....       3470      1384
```

Figure 7. Metrics of the logistic regression model

5.1.3 Decision Tree

Python created a decision tree with the maximum depth of the tree chose from the parameter list. The DecisionTreeClassifier from sklearn used the cleaned data set to fit the decision tree.

Parameters:

The function to measure the quality of a split is default as “gini”.

Set up a list of parameters for “max_depth” (3, 4, 5, 6, 7, 8, 10, 15, 25, 30, 35)

We set the minimum number of samples required to be at a leaf nodes and the minimum number of samples required to split an internal node to 5. Others use the default values.

Cross validation:

Evaluate and compare different models using 10-folds cross validation. Import cross_validate from sklearn.model_selection to calculate recall, accuracy , precision and F1 score.

After running the program, the results are shown in Figure 9.

The metrics created are based upon 10 folds, and we get the mean and standard deviation calculated from each of cv folds, each validation data consisting of 10% of the data randomly selected, in this case, it’s 3000 observations.

```
Maximum Tree Depth:  3
Metric.......  Mean    Std. Dev.
accuracy..... 0.8750    0.0068
recall....... 0.4156    0.0309
precision.... 0.6894    0.0402
f1........... 0.5178    0.0291

Maximum Tree Depth:  4
Metric.......  Mean    Std. Dev.
accuracy..... 0.8748    0.0077
recall....... 0.4565    0.0333
precision.... 0.6663    0.0408
f1........... 0.5410    0.0298

Maximum Tree Depth:  5
Metric.......  Mean    Std. Dev.
accuracy..... 0.8740    0.0071
recall....... 0.4477    0.0403
precision.... 0.6674    0.0449
f1........... 0.5342    0.0300

Maximum Tree Depth:  6
Metric.......  Mean    Std. Dev.
accuracy..... 0.8751    0.0067
recall....... 0.4516    0.0386
precision.... 0.6709    0.0372
f1........... 0.5386    0.0303

Maximum Tree Depth:  7
Metric.......  Mean    Std. Dev.
accuracy..... 0.8737    0.0065
recall....... 0.4442    0.0352
precision.... 0.6662    0.0372
f1........... 0.5317    0.0273

Maximum Tree Depth:  8
Metric.......  Mean    Std. Dev.
accuracy..... 0.8722    0.0069
recall....... 0.4427    0.0418
precision.... 0.6569    0.0370
f1........... 0.5277    0.0320

Maximum Tree Depth:  10
Metric.......  Mean    Std. Dev.
accuracy..... 0.8702    0.0080
recall....... 0.4434    0.0361
precision.... 0.6461    0.0456
f1........... 0.5246    0.0301

Maximum Tree Depth:  15
Metric.......  Mean    Std. Dev.
accuracy..... 0.8551    0.0098
recall....... 0.4481    0.0291
precision.... 0.5681    0.0429
f1........... 0.5002    0.0276

Maximum Tree Depth:  25
Metric.......  Mean    Std. Dev.
accuracy..... 0.8407    0.0104
recall....... 0.4541    0.0225
precision.... 0.5106    0.0379
f1........... 0.4801    0.0243

Maximum Tree Depth:  30
Metric.......  Mean    Std. Dev.
accuracy..... 0.8396    0.0102
recall....... 0.4565    0.0237
precision.... 0.5061    0.0366
f1........... 0.4796    0.0259

Maximum Tree Depth:  35
Metric.......  Mean    Std. Dev.
accuracy..... 0.8410    0.0098
recall....... 0.4578    0.0299
precision.... 0.5110    0.0353
f1........... 0.4823    0.0270
```

Figure 9. Decision tree metrics

Results:

From the metrics, as the tree depth increased, the measurements get worse, e.g. lower recall and F1 score. The best result comes from the model with maximum depth of 6 comparing to others with highest precision. Comparing to the model with tree depth 4, although the latter has a little bit higher recall, yet its precision is lower. Using a simple cross validation with training and validation part consisting of 70% and 30% of the data to compare these two models, the model have better classification result using the one with max depth set to 6.

5.1.4 Neural Network

For building neural network, we first create the features (X) and target (y) for building decision tree. Then we set network sizes of (3), (4), (5), (6), (7), (8), (9), (11) in one hidden layer and (3,2), (4,3), (5,4), (6,5), (7,6), (8,7), (9,8), (10,10) in two layers. We build a for loop to fit every network size for our neural network model by MLPClassifier from sklearn. At last, we calculate the metrics MISC, recall, accuracy, precision and F1 from the 10 cross-validation folds for each model to compare neural network with a different number of hidden layers and number of neurons and use Class_FNN to print the result, as shown following:

```
Network:  3
Metric.......  Mean     Std. Dev.
accuracy..... 0.8713    0.0069
recall....... 0.4073    0.0459
precision.... 0.6708    0.0434
f1........... 0.5048    0.0352

Network:  4
Metric.......  Mean     Std. Dev.
accuracy..... 0.8727    0.0087
recall....... 0.4415    0.0399
precision.... 0.6616    0.0480
f1........... 0.5284    0.0345

Network:  5
Metric.......  Mean     Std. Dev.
accuracy..... 0.8736    0.0074
recall....... 0.4463    0.0343
precision.... 0.6641    0.0409
f1........... 0.5329    0.0293

Network:  6
Metric.......  Mean     Std. Dev.
accuracy..... 0.8714    0.0075
recall....... 0.4275    0.0274
precision.... 0.6601    0.0438
f1........... 0.5181    0.0256

Network:  7
Metric.......  Mean     Std. Dev.
accuracy..... 0.8719    0.0076
recall....... 0.4347    0.0339
precision.... 0.6598    0.0452
f1........... 0.5230    0.0297

Network:  8
Metric.......  Mean     Std. Dev.
accuracy..... 0.8724    0.0077
recall....... 0.4407    0.0398
precision.... 0.6604    0.0443
f1........... 0.5272    0.0317

Network:  9
Metric.......  Mean     Std. Dev.
accuracy..... 0.8702    0.0085
recall....... 0.4343    0.0344
precision.... 0.6499    0.0482
f1........... 0.5195    0.0315

Network:  11
Metric.......  Mean     Std. Dev.
accuracy..... 0.8694    0.0073
recall....... 0.4417    0.0358
precision.... 0.6415    0.0389
f1........... 0.5221    0.0288

Network:  (3, 2)
Metric.......  Mean     Std. Dev.
accuracy..... 0.8730    0.0074
recall....... 0.4541    0.0375
precision.... 0.6573    0.0417
f1........... 0.5359    0.0297

Network:  (4, 3)
Metric.......  Mean     Std. Dev.
accuracy..... 0.8743    0.0072
recall....... 0.4384    0.0396
precision.... 0.6724    0.0421
f1........... 0.5295    0.0323

Network:  (5, 4)
Metric.......  Mean     Std. Dev.
accuracy..... 0.8720    0.0075
recall....... 0.4502    0.0366
precision.... 0.6536    0.0426
f1........... 0.5318    0.0283

Network:  (6, 5)
Metric.......  Mean     Std. Dev.
accuracy..... 0.8712    0.0065
recall....... 0.4429    0.0386
precision.... 0.6517    0.0364
f1........... 0.5261    0.0291

Network:  (7, 6)
Metric.......  Mean     Std. Dev.
accuracy..... 0.8711    0.0075
recall....... 0.4520    0.0290
precision.... 0.6465    0.0383
f1........... 0.5315    0.0274

Network:  (8, 7)
Metric.......  Mean     Std. Dev.
accuracy..... 0.8682    0.0071
recall....... 0.4413    0.0412
precision.... 0.6361    0.0431
f1........... 0.5194    0.0294

Network:  (9, 8)
Metric.......  Mean     Std. Dev.
accuracy..... 0.8668    0.0092
recall....... 0.4411    0.0264
precision.... 0.6270    0.0450
f1........... 0.5174    0.0294

Network:  (10, 10)
Metric.......  Mean     Std. Dev.
accuracy..... 0.8654    0.0080
recall....... 0.4343    0.0433
precision.... 0.6214    0.0401
f1........... 0.5099    0.0344
```

Figure 10. Neural network metrics

As shown in Figure 9, the best model with highest F1 score is the model with two hidden layers, the first layer with 3 perceptrons and the second layer with 2 perceptrons. Later, it will be used to compare with other model by using 70/30 training/validation split with calculating the same metrics MISC, recall, accuracy, precision and F1 from the validation results.

5.1.5 Random Forest

Use the RandomForestClassifier from sklearn module to construct a random forest solution. Select the parameters from the list of “n_estimators” (the number of trees constructed in the random forest) and the “max_features” (the maximum number of features allowed in each tree) shown below to build up different models.

Parameter:

- ➢ The list of the number of estimators is (27, 35, 45, 55, 60, 65, 70, 75, 80).
- ➢ The list of maximum features is ('auto', 0.3, 0.5, 0.8).
- ➢ The function to measure the quality of a split is set to ‘gini’.
- ➢ The maximum depth of the tree is default as None, which means nodes are expanded until all leaves are expanded until all leaves contain less than min_samples_split samples.

In this case, the minimum samples split is set to 2.

Metrics displaying the cross-validation results:

```
Best based on F1-Score
Best Number of Estimators (trees) =  75
Best Maximum Features =  0.8
```

Figure 11. Best random forest model

Result:

The best solution is defined by the one with maximum F1-Score and the best result can be produced from the code.

From the 10-fold cross-validation it appears that the solution using 75 trees and 80% features produced the highest F1-score. In this case, there are 41 features including the encoding nominal variables, so it means our model allows for no more than 33 features in each tree.

5.1.6 70/30 Model Comparison

After we fit the data set with 4 different models (logistic regression, decision tree, neural network and decision tree, we need compare these four solutions and select the best one. In this case, we use 70/30 training/validation split to fit each solution with 70% data set and calculate the same metrics MISC, recall, accuracy, precision and F1 from the validation results, shown as following:

Logistic Regression:

```
Training Data
Random Selection of 70% of Original Data


Model Metrics..........         Training    Validation
Observations...........            21001          9001
Coefficients...........               42            42
DF Error...............            20959          8959
Mean Absolute Error....           0.2099        0.2094
Avg Squared Error......           0.1035        0.1036
Accuracy...............           0.8630        0.8643
Precision..............           0.6811        0.6657
Recall (Sensitivity)...           0.2958        0.3044
F1-score...............           0.4124        0.4177
```

Figure 12. Logistic regression 70/30 metrics

Decision Tree: (max_depth = 6)

```
Model Metrics..........         Training    Validation
Observations...........            21001          9001
Features...............               41            41
Maximum Tree Depth.....                6             6
Minimum Leaf Size......                5             5
Minimum split Size.....                5             5
Mean Absolute Error....           0.1856        0.1903
Avg Squared Error......           0.0928        0.0971
Accuracy...............           0.8811        0.8756
Precision..............           0.6911        0.6547
Recall (Sensitivity)...           0.4855        0.4691
F1-score...............           0.5703        0.5466
MISC (Misclassification)...        11.9%         12.4%
     class 0...............         4.2%          4.7%
     class 1...............        51.4%         53.1%


Training
Confusion Matrix  Class 0   Class 1
Class 0.....      16845      741
Class 1.....       1757     1658


Validation
Confusion Matrix  Class 0   Class 1
Class 0.....       7206      356
Class 1.....        764      675
```

Figure 13. Decision tree 70/30 metrics

Neural Network:

```
******** NEURAL NETWORK ********

Model Metrics
Observations...........        21001
Features...............           41
Number of Layers.......            2
Number of Outputs......            1
Number of Weights......          137
Activation Function....     logistic
Loss...................       0.0476
R-Squared..............       0.3010
Mean Absolute Error....       0.1905
Median Absolute Error..       0.0761
Avg Squared Error......       0.0952
Square Root ASE........       0.3085
```

Figure 14. Neural network 70/30 metrics

Random Forest: (n_estimator = 75, max_features = 0.8)

```
 Random forest: Training Data
Random Selection of 70% of Original Data

Model Metrics..........      Training    Validation
Observations...........         21001          9001
Features...............            41            41
Maximum Tree Depth.....          None          None
Minimum Leaf Size......             1             1
Minimum split Size.....             2             2
Mean Absolute Error....        0.0695        0.1883
Avg Squared Error......        0.0130        0.0907
Accuracy...............        0.9999        0.8835
Precision..............        1.0000        0.6958
Recall (Sensitivity)...        0.9994        0.4816
F1-score...............        0.9997        0.5692
MISC (Misclassification)...      0.0%         11.7%
     class 0..............       0.0%          4.0%
     class 1..............       0.1%         51.8%

Training
Confusion Matrix  Class 0   Class 1
Class 0.....      17586        0
Class 1.....          2     3413

Validation
Confusion Matrix  Class 0   Class 1
Class 0.....       7259      303
Class 1.....        746      693
```

```
FEATURE.........  IMPORTANC
Jun_Status......   0.2016
May_Status......   0.0596
Age.............   0.0520
Jun_Bill........   0.0457
Credit_Limit....   0.0448
Jan_Payment.....   0.0342
Jan_Bill........   0.0322
May_Bill........   0.0315
May_Payment.....   0.0310
Apr_Payment.....   0.0307
Mar_Bill........   0.0304
May_PayPercent..   0.0301
Feb_Bill........   0.0296
Jun_Payment.....   0.0296
Apr_PayPercent..   0.0293
Mar_PayPercent..   0.0291
Mar_Payment.....   0.0290
Apr_Bill........   0.0289
Feb_Payment.....   0.0283
Feb_PayPercent..   0.0282
Jun_PayPercent..   0.0279
Jan_PayPercent..   0.0272
Mar_Status......   0.0130
Apr_Status......   0.0119
Jan_Status......   0.0110
Feb_Status......   0.0095
Default.........   0.0077
Education2......   0.0058
Education1......   0.0050
Marital_Status2.   0.0050
Marital_Status1.   0.0049
Education3......   0.0047
card_class1.....   0.0034
Marital_Status3.   0.0021
card_class0.....   0.0019
card_class2.....   0.0013
Education5......   0.0008
Education6......   0.0004
Marital_Status0.   0.0004
Education4......   0.0003
Education0......   0.0000
```

Figure 15. Random Forest 70/30 metrics

From the simple cross validation (70/30), the logistic regression, decision tree and neutral networks fit to these data using our parameter settings have a much higher misclassification rate than the random forest. It also has much higher F1 score than others. Therefore, we conclude that the random forest is the best model among four, even though all of these F1 scores are not so good. It is possible that we need do more data processing before create the model, like feature selection.

6. Reflection & Improvement

Up to now, we have already got the best models. Obviously, the random forest performs best in Python on the validation data, except a lower specificity. As we have discussed before, what we care most in this problem is the recall rate, so that we can choose the model that is able to find as many potential default behaviors as possible.

Two possible reasons might limit the performance of random forest in Python. The first one is overfitting. As we can see from the confusion matrices of training and validation data in 70/30 split, the false positive and false negative number are almost zero. With the training set fit being overwhelmingly good, the F1 score of the validation drops drastically below 0.6. This is typical overfitting, usually caused by fitting too many variables in the model. Secondly, we still need variable selection before running the model. By looking at the names of all the attributes, we should notice the potential collinearity among them. For example, as a function of payment divided by bill, monthly paypercent should have some correlation with those two kinds of attributes. Therefore, dropping them from the very beginning is probably a good way to simplify our model and make it more explainable.

However, the metrics are not good enough for prediction or further application. We tried to drop some variables due to Gini reduction before running the random forest, but the results were still not good enough. More effective variable selection is needed before fitting any model for the dataset. Besides, when the model metrics like F1 score can’t meet our requirement, we should also consider adjusting criterion for choosing the best model. Using ROC curve in this case is a more robust way to judge the performance of different models.